Compilers and Formal Languages

Email:	christian.urban at kcl.ac.uk
Office Hour:	Thurdays 15 – 16
Location:	N7.07 (North Wing, Bush House)
Slides & Progs:	KEATS
Pollev:	<pre>https://pollev.com/cfltutoratki576</pre>

1 Introduction, Languages	6 While-Language
2 Regular Expressions, Derivatives	7 Compilation, JVM
3 Automata, Regular Languages	8 Compiling Functional Languages
4 Lexing, Tokenising	9 Optimisations
5 Grammars, Parsing	10 LLVM

The Fun Language

```
def fact(n) = if n == 0 then 1 else n * fact(n - 1);
```

def gcd(a, b) = if b == 0 then a else gcd(b, a % b);

Factorial

```
.method public static fact(I)I
.limit locals 1
.limit stack 6
  iload 0
  1dc 0
  if icmpne If else 0
  ldc 1
  goto If end 1
                             def fact(n) =
If else 0:
                               if n == 0 then 1
  iload 0
  iload 0
                               else n * fact(n - 1)
  1dc 1
  isub
  invokestatic fact/fact(I)I
  imul
If end 1:
  ireturn
.end method
```

```
.method public static facT(II)I Factorial
limit locals 2
.limit stack 6
 iload 0
 1dc 0
 if_icmpne If_else 2
 iload 1
 goto If end 3
If else 2:
                            def facT(n, acc) =
 iload 0
                               if n == 0 then acc
 ldc 1
                               else facT(n - 1, n * acc)
 isub
 iload 0
 iload 1
 imul
 invokestatic fact/fact/facT(II)I
If end 3:
 ireturn
.end method
```

.method public static +	facT(II)I
.limit locals 2	
.limit stack 6	
<pre>facT_Start:</pre>	
iload 0	
ldc 0	
<pre>if_icmpne If_else_2</pre>	
iload 1	
goto If_end_3	
If_else_2:	
iload 0	def facT(n, acc) =
ldc 1	if n == 0 then acc
isub	else facT(n - 1, n * acc);
iload 0	
iload 1	
imul	
istore 1	
istore 0	
goto facT_Start	

If_end_3:

CFL 09, King's College London – p. 5/40

Tail Recursion

A call to f(args) is usually compiled as

```
args onto stack invokestatic .../f
```

Tail Recursion

A call to f(args) is usually compiled as

```
args onto stack
invokestatic .../f
```

A call is in tail position provided:

• if Bexp then Exp else Exp



- Exp ; Exp
- Exp op Exp then a call f(args) can be compiled as

```
prepare environment
jump to start of function
```

Tail Recursive Call

```
def compile expT(a: Exp, env: Mem, name: String): Instrs =
  . . .
  case Call(n, args) => if (name == n)
    val stores =
      args.zipWithIndex.map { case (x, y) => i"istore $y" }
    args.map(a => compile expT(a, env, "")).mkString ++
    stores.reverse.mkString ++
    i"goto ${n} Start"
  } else {
    val is = "I" * args.length
    args.map(a => compile_expT(a, env, "")).mkString ++
    i"invokestatic XXX/XXX/${n}(${is})I"
```

???

	REM POPPINS SOUND FOR S-15 TO 8 STEP -1			1 4010	PM+PB-38:0=P1+12:F=D +3:0T=P0-10:P0=Pm+23
	1,14,0,5:NEXT 5 SOUND 0,0,0,0:SOUND 1	E ·		2 4020	180NUE-1800 08APHICS 18:POKE 708 ,134:POKE 709,198:PO KE 710,84:POKE 754,C
		2			KE 718,841PDKE 754,0
		12		11 4030	
	PEK+PEEK(53261)+1P P EK+8 THEN RETURN	8	7		ALLION CRAIN! POBIT
121002	FOR 1+25 TO 18 STEP		±	11 40 40	DIN SHILDIISSTOR (De
	-5:800ND 8,1,4,8:800 ND 1,1+2,2,8:NEXT 1:		LEVELIE		30420"+FOR 1=8 TO 18
	SCUND 0.0.0.SCUND 1.0.0.0.RETURN				
		Player/	missile graphics enimate the red balloons in the Atari persian	14050	.1017 861561NEXT 1 1F PEEK(55279)()6 TH
	HEAD? IF BY-DB(3 THEN PRAC		con Craty."		EN 4858 RETURN
	H+DE, P#+BE) =B8:88+38 -11:86L=84L-1:80H+80				POKE 53248. P. POKE 53
			IF BCOBONUE THEN LF= LF+11BONUE+BONUE+120		244, BIGRAPHICS 171PD
1.1019	R/311POP 100TO 68 REM DID ANM HIT BALL		##LVLICOLDR 721PLDT		198: PEKE 710.84 POSITION 6.417 #41"0
11820	GON7 IF STRIG(#) =# THEN 1	2 2040	LF.23 IF BOHCL THEN 2060		
~ 1020	# ##\$(#Y-178)<3 THEN	0 2450	FOR FEL TO BOHIFOR J	1 4120	
11 1030	1040 A+&+((A)PP)-(A(+PP))		LIPHS(PD+J,PD+J+121= BAINERT JINERT KINER		
	*SIPOKE S3249, AIPOKE PC, LIRETURN		PHE (PT, PR) -NE . RETURN		GN 4.1217 461 10 8 38
	PC, LINETURN PAAIIF HIT THEN PAPP +((P)PP)-(P(*PP))#3			10.4150	IF PEEK (53279) ()& TH EN 4120
		13 2499	REM CHODSE BALLOON T	10.41.40	
	-31608U8 500:800N0 0 .E.10.81PH8(PL+K,F+K	N 2560	BR=3+INT(RND(0)#41#3		REN REDEFINE CHARACT
	1=851PGKE 53249,P P#P+(((P)PP)-(P(#PP)			11.4500	
			R, I, BT: IF BT-32 THEN NEXT 1:GOTO 2500	#4510	
	PEEK(53241)=#))))PORE	11 2510	PENE 705, PEEX(707+E) 1-1, 80/3111A+248(80/ 3+2118L+32+188+POLE		61CH0+57544 1F PEEKICH5+9100 TH
18 1070	PC.1 NEXT KIDSUND 0.0.0.0		3+2):8L=32+1#8:POLE 53249,6	8 4520	EN RETURN FOR 1=0 TO SILLPONE
10 1 4 9 9	NEM MAN MISSED BALLO	11 2524	PMSIPI-BL, D-BL) -BSIC OLOR 32:PLOT BR, 1	H 4510	
** 1500	PR&(P1+BT, P1+BY+100)	1 2570		0.4540	AESTORE 4540-FOR 1-1
H 1510	PRAIRM, PRINER(1, 371)		ISL=INT(ILVL/3+11/2+ BS1+RETURN		TO 21CHP=CHS+(ABC(C +(1))-32)+8+POR 2+0
	PR& (P#+198, PB) - MGR41 POKE 53258, PP-0(POK 1+288 TO 8 STEP -181	4.3000	REN DRAW SCREEN RESTORE SPERIDIM F(5		TO TIREAD AIPDLE ENP +J, AINEET JINEET I
	1-200 TO 0 STEP -101 BOUND 0.1.7.10:NETT		1,014,81,84L1611FOR 1=1 TO 31READ AIF(1)	17 4558	
			*AINEXT IIDATA #, 32,		CHS+1,255-PEEX(CHD+1 DINEXT LIBETURN
* 152#	SOUND 0,0,0,011F BON (1 THEN 1550 FOR 1=158 TO BE STEP	1.3010	FF=120.PONE 53240,PP	11 4560	8074 28,58,125,125,1 21,62,20,0 8074 8,16,56,124,56,
			IBAL=24:LVL=LVL=1:1P LVL>10 THEN LVL=10		
01540	PH&(P#+J,P#+J+12)=14 (NEXT J1PH&(P#+J,P#+	1 3020	*081110W 11,2317 #61 *0805898*1LVL1F08 1=1	04999	REM STE UP P/M
	Jail2)=POStidosus are		TO LFICOLOR 721PLOT		REH STE UP P/H DIM PH8(4874), P\$(255 1,N8(12), B8(15), P088 (12), M058(16), MD88(1
	1PH8(P0+3,P0+3+12)=N	0.3030	FOR X=3 TO 18 STEP 3		61,H\$122) FOR 1=1 TO 12(N\$(1)=
K 155e	FOR I=1 TO LODINEX! IIFOKE 53250, PICOLON				
			*FIAISC(Y-1,X/3) #AIP LOT X, VINERT VINERT	11 5020	A=ADR(PHE)(PHE=INT(A)/2048)#2048;1F PHE-0
1.1540	0.0.0.0 PHS(PT,PB)-PSIPOKE 5	11 3540	TARETURN		THEN PHE-PHE-2040
	3248, PPIBDN+8:88=149 IFF=128:RETURN REN PDP BALLODNS AND		ON SERTITIONE TRE, 1 34+POKE TRT, 198+POKE	17 3.04.0	H8/2561POKE 53277,3 PH8-CHR8(0)1PH8(4096
15 2888	IF BOHAL THEN RETURN	- 3518	FOR 1=4 TO 16: POSITI ON 1,117 86178(1-3,1	1 3454	1P8-PR8 P8-5-10241POXE 784.5
1 2010	FOR 1=1 TO BOHLPHS (P	11 3524	-3) INEST 1 POSITION 1.017 #61"8	12 3 6 4 8	41RESTORE 5070 FDR 1=100 TO 137:000
	H+1, PD+1)=H8 PM8(P#+158, PM)=PD89:	5 3520	COPPI'ISC RETURN		D AIPS(1)+CHRE(A) INE XT IIPHE(PD+A7, PD+20
	BOSUB ADDIPOR V=1 TO 15INERT VIPHS(PD+15	11 3 9 9 9	AND STARTING DISPLA	11 5.070	
				1.5078	
	H& (PH, PB) +P\$ (100,137		PP=120:80=169:5C=01L VL=01LF=3:FC=53278:H		0,50,10,10,55,55,124 ,124,106,105,105,105
# 2020	SC-SC+BAL(1)#5#LVLIP GBITION 7,017 #615C1		*P8-121PT=F8+691PB+P 8+285		.120,120,40,40,40,40
COMPU					

	BE 18 72		77 07781 0 78 07861 0	0 A0 82 80 8		04301 02 E6 17 46 75 80 DE 93 04301 00 D0 E0 60 60 60 07 04
84081	0E CA 34		92 @?@@i F	C AV BC 85 P	DA 08 08 FA 0	BA481 48 81 FC AA 48 10 08 61
	51 BE CN			E BD A9 #2 E		BA48, 40 70 80 51 FE 91 FE 80 50 FE
046.01	0E 0D 4E				2 98 17 28 82	84581 08 67 CB CC 6F ED D8 E8 84581 45 70 ED 51 FE V1 FE A8
04701	14 05 FC	AV BC BS FD AN		# AC 73 00 C		
	00 00 12	BD M9 15 80 44		5 A0 00 EE 7 0 A7 00 80 7		04401 85 40 74 85 85 FF 40 71 94701 85 27 85 FJ 42 42 64 18
05401	77 80 85	73 80 49 90 86	74 07C#1 0	0 10 10 09 5		BATH: 85 29 65 FJ 22 69 DJ 18 BATH: 25 71 80 26 24 18 64 19
85184	71 85 45	## 60 7# 60 2#	05 8708+ 8		1 60 AT OF 50	
	17 09 28	53 07 28 08 8F	7D 87D#+ 8	0 4E 00 2C 0		04031 40 00 04 08 0C 10 14 18
05201	DP FA AZ	## HD 4# ME 11	TO B750 0			BAYRA IC ## #4 #0 #C 1# 14 18 BAYRA IC #1 #5 #2 #5 11 15 19
05301	40 OE EB	EC 07 00 0# F4	M 87EB: F	0 07 40 47 0	00 BF 60 5C	8448: 10 01 05 07 00 11 15 19 8440: 10 01 05 07 00 11 15 19
	CE 07 00	20 62 09 20 EE 89 20 CA 86 4C	0 87F01 A			BAART 10 07 05 07 00 11 15 17
	92 84 65	92 80 89 80 62	C 86001 7			04801 1E 02 05 04 0E 12 15 14
	AN DE DO	80 AE 88 80 EC			8 60 40 80 CC	0480; 16 03 07 08 09 13 17 18 0400; 17 03 07 08 09 13 17 18
	87 80 74	29 80 45 86 80	17 00101 0			
	45 77 86	E0 73 83 40 ES	17 BE201 D			BAD#1 1# #1 FF F7 58 C6 D4 A2
	00 00 71	ED //Y B# BD B3	M 8828+ 6			04001 90 00 40 49 FF 00 19 00 04001 40 FD 04 49 70 00 19 00
	80 24 85	87 AG 07 80 38 87 80 EE 88 80	2 88301 A			BAER, 4C FR DA 07 71 ED 19 88
10.000	40 54 85	48 40 87 86 ED	5 00301 C			BAF#1 4C FB BA A9 72 80 19 88
85701	01 40 AT	#0 00 03 00 AV	17 8848. P			BAPB: 40 FB BA A7 81 80 53 88 8008: 48 88 67 58 80 20 00 47
	05 00 0A	60 AD 88 80 CD	10 88541 A			
	80 CD 77	BD BA LE AD BA		A 40 92 80 C	07 98 97 88	
85641	80 49 #6	CV 87 98 85 EV				8810; ## 7# C0 80 54 80 AV 88 8829; 80 53 80 46 54 88 70 83
	07 EE 88	80 00 0A 00 AD 40 40 40 00 00 00 00 00 00	C 08761 V			
05001	C9 24 D#	DØ AD BA BD 38		2 60 A3 92 8	0 00 73 00 24	88381 98 #3 40 38 C8 42 FF ED
05501	EV 03 80	#5 AV #7 CE 88				60301 D# FD EE 55 08 D# D3 10
	80 80 84	80 A0 88 80 38 89 80 38 80 76	27 00741 F			
01201	00 00 78	ED 01 01 05 05 30				8850+ VØ BA AØ 20 2E 20 80 BE
05701	ED 78 80	18 40 BA 80 ED	2			8858: 03 08 FT 01 08 AT 08 00 8848: ET 08 08 FE 08 08 50 00
	77 ED ED	00 00 10 05 47	D			ARAE: DO DE DA 40 10 DA 80 L1
	80 ED 67	00 10 69 51 00 70 00 00 00 47	E 88001 F			887#+ 81 90 E4 81 8E 01 83 86
		60 AD SE 80 80		0 F5 AD 14 C		88781 C4 03 05 71 07 05 04 07 08097 86 71 06 CC AA DC D2 AA
	BY BO AL	08 80 8C 07 85	8 8804+ C			
	EØ #5 40 80 7E 80	AD 76 00 00 72	C 88001 C			
		80 80 73 80 A5				0848+ 10 A0 01 00 A0 01 F0 23 0840+ 07 FC FF 97 F0 00 07 F0
	87 80 C4	52 10 35 43 88				
	ED 28 E0 EC ED 74	33 18 05 47 77	6 88F81 C			
	30 64 04	18 40 73 80 34	Y 9766; 6			88881 88 C8 97 88 88 88 88 A8 88 88 A8
	PC 07 07	90 10 E9 07 EE	10 BV141 P	0 20 AS 98 A		00C0: 50 71 DC 00 04 DC 7C 71
	72 80 4C CE 72 80	70 17 00 73 00	17 BV18: 2			00041 0C 1C 04 0C 0E 11 0E CA
	4C 77 84	60 08 80 80 72		0 17 04 05 0	14 04 45 57	09001 AA DA C3 AA 01 C0 AA 04 04 04
	80 AD DA	00 00 73 00 AD			12 CT FT AT	08081 80 D0 A0 81 D0 A0 81 D0
	89 80 CV	04 90 04 ME 83	2 89381 P			0074: A0 01 FC F8 01 FF FF 07
	80 71 80	A1 84 80 81 88	0 0740. 4	C #7 84 D# C		08781 90 E0 01 90 E0 01 00 20 00 00 00 00 00 00 00 00 00 00 00
	20 05 07	A5 57 50 59 EV	F 89561 C			OC481 68 87 55 88 88 5F 88 88
	PE ED 07	00 EE 10 00 4C	23 89501 D			01101 50 00 00 70 00 00 01 00 01101 00 03 07 07 03 07 00
	RD 88: 893	CT ED 50 ET 22		0 49 95 60 0	0 02 00 FD DE	8C18+ P6 83 87 87 87 83 87 86
	53 87 24	42 07 28 EE 07	W 87701 0			8028+ PE 05 03 C# 44 00 D# AF
	28 62 09	20 CA 05 40 PF	1 89701 7			0C34: 81 P4 48 85 P4 44 85 84 9C38: 46 85 80 44 85 84 45 85
BADNO -	00 77 80	80 73 80 20 69		2 80 91 18 0	1 AD 73 80 FO	
	00 14 49	M7 AA 95 FC M7				
	10 05 FD	4C EE 86 AV 56	14 8110. A		0 00 91 1E DA	9054: 80 88 08 88 88 88 88 88 88 8058: 04 68 81 74 78 85 04 68
	85 FC AT	00 05 FD 67 02		1 1E C0 AD 6 0 00 7# 80 9	0 00 41 1E DA	8C58+ D4 A8 81 F4 F8 85 04 A8
	AT 40 ED	71 00 47 00 02	C 89941 1	0 49 00 05 1	98 92 66 61	SCAR: 00 01 04 00 01 04 44 50
	78 80 28	77 BY AD 6A 80	PC 8180/ 1			BC781 CO AA BI CO AA BI CO BA
	FØ 3E CE	6A 80 AD 76 80	10 89C81 7 89C81 8			SC7D: DJ CJ AA DJ DJ BA DJ DC
	60 72 8D 80 20 49	AD 77 80 80 73	a 0100 0	C AF 80 28 6	# 04 2# 3C 63	SCERI TO TO BE AC BC FA BE BC FA
	00 2C 44	BC 05 FD 4C 31	81081 8	A EE 71 80 A		BCYD: FE BC ES BD ED ES BC BD
	97 AT FE	65 FC AT 60 85				8C901 00 04 FE DO DO FE 05 DE
	FD A4 #2	80 40 80 AV 84	84 07ED: C			SCAP, EP EA SC EC BA RE EA EA SCAD, BC PE EP DO PR SC RC EC
	00 AE ED	AV 85 80 71 80	LA 01701 0			SCADY BC PE EP DO VS SC SC EC
		PA PE 38 65 55	1.5	0 72 80 08 8	1 15 60 73 48	
877581	83 45 60	AD 74 80 C7 48	44. EN-001 0		5 FC C8 81 91	BELDI EE AV 68 05 ED AV 68 05
07581	AT 20 20	AA 80 54 C2 A3	16 0A101 1 87 0A101 0			BCCB1 EF A0 E0 BC 73 80 AV 00
	74 80 85	24 30 67 08 40		1 1E ED 7# 0	20 50 97 97 44	BCDB1 23 ED FE do do 26 1E CO.
87741	8D 83 80	AD 77 80 89 71	72 84281 8	5 1E 18 AT 2		BCK#1 D# FA 2C 7# 80 1# #2 38

Opcodes

84CØ: 8E 18 7D 51 BE 9D 51 BE F7 84C8: C9 ØA 9Ø 12 E9 ØA 9D 51 70 84DØ: ØA FE BE CA 30 51 BE 92 BD B4DB: 51 BE C9 ØA BØ EE 20 EB 75 84EØ1 87 AD 4D BE 09 8D BØ 4D 5E 84E8: BE BD 4D BE 20 4A 85 A9 0E 84FØ: 16 85 FC A9 8C 85 FD A9 CØ 84F8: 00 8D 6D 8D A9 15 8D 6E 17

CFL 09, King's College London – p. 10/40

CFL 09, King's College London – p. 10/40

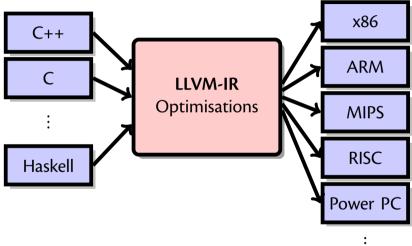
Peephole Optimisations

- ldc: iconst_0...iconst_5 bipush n where -128 < n <= 128
- iload: iload_0...iload_3
- istore: istore_0...istore_3



- Chris Lattner, Vikram Adve (started in 2000)
- Apple hired Lattner in 2006
- modular architecture, LLVM-IR
- lli and llc

LLVM: Overview



LLVM-IR

```
define i32 @fact (i32 %n) {
   %tmp 19 = icmp eq i32 %n, 0
   br i1 %tmp 19, label %if br 23, label %else br 24
if br 23:
   ret i32 1
else br 24:
   %tmp 21 = sub i32 %n, 1
   %tmp 22 = call i32 @fact (i32 %tmp 21)
   %tmp 20 = mul i32 %n, %tmp 22
   ret i32 %tmp 20
                                    def fact(n) = 
                                      if n == 0 then 1
                                      else n * fact(n - 1)
```

LLVM Types

boolean	i1
byte	i8
short	i16
char	i16
integer	i32
long	i64
float	float
double	double
*	pointer to
**	pointer to a pointer to
[_]	arrays of

LLVM-IR Instructions

br i1 %var, label %if_br, label %else_br

icmp eq i32 %x, %y ; for equal icmp sle i32 %x, %y ; signed less or equal icmp slt i32 %x, %y ; signed less than icmp ult i32 %x, %y ; unsigned less than

%var = call i32 @foo(...args...)

SSA Format

(1+a) + (3 + (b * 5))

tmp0 = <mark>add</mark>	1 a
tmp1 = mul	b 5
tmp2 = add	3 tmp1
tmp3 = add	tmp0 tmp2

Static Single Assignment

Abstract Syntax Trees

// Fun language (expressions)
abstract class Exp
abstract class BExp

case class Call(name: String, args: List[Exp]) extends Exp case class If(a: BExp, e1: Exp, e2: Exp) extends Exp case class Write(e: Exp) extends Exp case class Var(s: String) extends Exp case class Num(i: Int) extends Exp case class Aop(o: String, a1: Exp, a2: Exp) extends Exp case class Sequence(e1: Exp, e2: Exp) extends Exp case class Bop(o: String, a1: Exp, a2: Exp) extends BExp

K-(Intermediate)Language

abstract class KExp abstract class KVal

// K-Values
case class KVar(s: String) extends KVal
case class KNum(i: Int) extends KVal
case class Kop(o: String, v1: KVal, v2: KVal) extends KVal
case class KCall(o: String, vrs: List[KVal]) extends KVal
case class KWrite(v: KVal) extends KVal

// K-Expressions

case class KIf(x1: String, e1: KExp, e2: KExp) extends KExp
case class KLet(x: String, v: KVal, e: KExp) extends KExp
case class KReturn(v: KVal) extends KExp

KLet

```
tmp0 = add 1 a
tmp1 = mul b 5
tmp2 = add 3 tmp1
tmp3 = add tmp0 tmp2
 KLet tmp0 , add 1 a in
  KLet tmp1 , mul b 5 in
   KLet tmp2 , add 3 tmp1 in
     KLet tmp3 , add tmp0 tmp2 in
```

• • •

case class KLet(x: String, e1: KVal, e2: KExp)

KLet

tmp0	=	add	1	а
tmp1	=	mul	b	5
tmp2	=	add	3	tmp1
tmp3	=	add	tn	np0 tmp2
let	t t	:mp0	=	add 1 a in
<pre>let tmp1 = mul b 5 in</pre>				
]	Let	t mp	52	<pre>= add 3 tmp1 in</pre>

let tmp3 = add tmp0 tmp2 in

• • •

case class KLet(x: String, e1: KVal, e2: KExp)

```
def CPS(e: Exp)(k: KVal => KExp) : KExp =
  e match { ... }
```

the continuation k can be thought of:

```
let tmp0 = add 1 a in
let tmp1 = mul 
    5 in
let tmp2 = add 3 tmp1 in
let tmp3 = add tmp0 tmp2 in
    KReturn tmp3
```

```
def CPS(e: Exp)(k: KVal => KExp) : KExp =
    e match {
        case Var(s) => k(KVar(s))
        case Num(i) => k(KNum(i))
        ...
    }
```

```
let tmp0 = add 1 a in
let tmp1 = mul 
    5 in
let tmp2 = add 3 tmp1 in
let tmp3 = add tmp0 tmp2 in
    KReturn tmp3
```

```
def CPS(e: Exp)(k: KVal => KExp) : KExp = e match {
  case Aop(o, e1, e2) => {
    val z = Fresh("tmp")
    CPS(e1)(y1 =>
      CPS(e2)(y2 =>
                KLet(z, Kop(o, y1, y2), k(KVar(z))))
  } ...
                 . . .
                 let z = op \square_{v_1} \square_{v_2}
                 let tmp0 = add 1 a in
                 let tmp1 = mul Z 5 in
                 let tmp2 = add 3 tmp1 in
                 let tmp3 = add tmp0 tmp2 in
                   KReturn tmp3
```

```
def CPS(e: Exp)(k: KVal => KExp) : KExp =
    e match {
        case Sequence(e1, e2) =>
            CPS(e1)(_ => CPS(e2)(y2 => k(y2)))
        ...
    }
```

```
let tmp0 = add 1 a in
let tmp1 = mul 
    5 in
let tmp2 = add 3 tmp1 in
let tmp3 = add tmp0 tmp2 in
    KReturn tmp3
```

```
def CPS(e: Exp)(k: KVal => KExp) : KExp =
  e match {
    . . .
    case If(Bop(o, b1, b2), e1, e2) => {
      val z = Fresh("tmp")
      CPS(b1)(v1 = >
        CPS(b2)(y2 =>
          KLet(z, Kop(o, y1, y2))
                 KIf(z, CPS(e1)(k), CPS(e2)(k))))
     }
    . . .
```

The Basic Language, 1980+

```
5 \text{ LET } S = 0
```

```
10 INPUT V
```

```
20 PRINT "Input number"
```

```
30 IF N = 0 THEN GOTO 99
```

```
40 \text{ FOR } I = 1 \text{ TO } N
```

```
45 LET S = S + V(I)
```

```
50 NEXT I
```

```
60 PRINT S/N
```

```
70 GOTO 5
```

```
99 END
```

"Spaghetti Code"

Target Specific ASM

llc -march=x86-64 fact.ll
llc -march=arm fact.ll

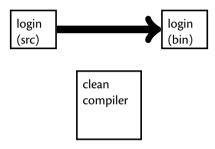
Intel: xorl %eax, %eax ARM: mov pc, lr

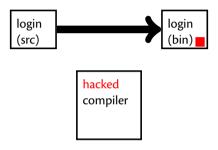
Using a compiler, how can you mount the perfect attack against a system?

What is a **perfect** attack?

- 1. you can potentially completely take over a target system
- 2. your attack is (nearly) undetectable
- 3. the victim has (almost) no chance to recover

clean compiler



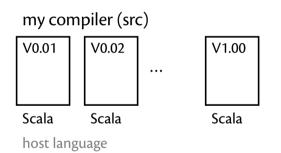


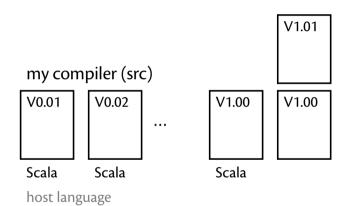
my compiler (src)

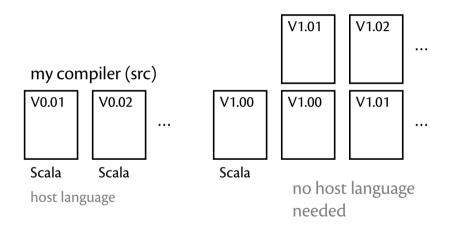
V0.01

Scala

host language







Hacking Compilers



Ken Thompson Turing Award, 1983

Ken Thompson showed how to hide a Trojan Horse in a compiler without leaving any traces in the source code. No amount of source level verification will protect you from such Thompson-hacks.

Hacking Compilers



Ken Thompson Turing Award, 19



- 1) Assume you ship the compiler as binary and also with sources.
- 2) Make the compiler aware when it compiles itself.
- 3) Add the Trojan horse.
- 4) Compile.

;0)

- 5) Delete Trojan horse from the sources of the compiler.
- 6) Go on holiday for the rest of your life.

ng any on will

acks.

a Tro-

Hacking Compilers



Ken Thompson Turing Award, 1983

Ken Thompson showed how to hide a Trojan Horse in a compiler without leaving any traces in the source code. No amount of source level verification will protect you from such Thompson-hacks.

Dijkstra on Testing

"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."

What is good about compilers: the either seem to work, or go horribly wrong (most of the time).

Proving Programs to be Correct

Theorem: There are infinitely many prime numbers.

Proof ...

similarly

Theorem: The program is doing what it is supposed to be doing.

Long, long proof ...

This can be a gigantic proof. The only hope is to have help from the computer. 'Program' is here to be understood to be quite general (compiler, OS, ...).

Can This Be Done?

- in 2008, verification of a small C-compiler
 - "if my input program has a certain behaviour, then the compiled machine code has the same behaviour"
 - is as good as gcc -01, but much, much less buggy



Fuzzy Testing C-Compilers

- tested GCC, LLVM and others by randomly generating C-programs
- found more than 300 bugs in GCC and also many in LLVM (some of them highest-level critical)
- about CompCert:

"The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task."



- Revision Lecture
- How many strings are in $L(a^*)$?



- Revision Lecture
- How many strings are in $L(a^*)$?
- How many strings are in L((a + b)*)?
 Are there more than in L(a*)?